What is claimed is:

1. A processor comprising

a first pipeline configured to execute essential code;

a second pipeline configured to execute non-essential code; and

a conjugate mapping table configured to specify non-essential code to be executed by the second pipeline.

2. The processor of claim 1 wherein the first pipeline is coupled to a first instruction cache configured to cache instructions that determine the logical correctness of a program.

3. The processor of claim 2 wherein the second pipeline is coupled to a second instruction cache configured to cache instructions that provide hints for the execution of the instructions that determine the logical correctness of the program.

4. The processor of claim 1 wherein the first pipeline is coupled to registers that store a microarchitectural state, and wherein the conjugate mapping table is responsive to the microarchitectural state.

5. The processor of claim 4 further comprising:

a first instruction cache coupled to the first pipeline; and

a second instruction cache coupled between the conjugate mapping table and the second pipeline.

6. The processor of claim 1 wherein the conjugate mapping table is responsive to a trigger, the trigger being mapped within the conjugate mapping table to the non-essential code.

7. The processor of claim 1 wherein the conjugate mapping table comprises a plurality of records, each of the plurality of records being configured to map a trigger to a non-essential code sequence.

8. The processor of claim 7 wherein the trigger comprises an atomic value, such that the conjugate mapping table is configured to specify the non-essential code sequence when the atomic value is satisfied.

9. The processor of claim 7 wherein the trigger comprises a vector value, such that the conjugate mapping table is configured to specify the non-essential code sequence when the vector value is satisfied.

10. The processor of claim 1 further comprising a microarchitectural structure coupled between the first pipeline and the second pipeline.

11. The processor of claim 10 wherein the microarchitectural structure comprises a register bank.

12. A processor comprising:

an instruction set architecture (ISA) visible path; and

a conjugate pipeline coupled to the ISA visible path, the conjugate pipeline being configured to execute hint calculus code and to provide execution hints to the ISA visible path.

13. The processor of claim 12 wherein the processor can take on architectural states and microarchitectural states, the conjugate pipeline being configured to affect microarchitectural states.

14. The processor of claim 12 wherein the ISA visible path includes a pipeline configured to execute instructions from a user program.

15. The processor of claim 14 wherein the conjugate pipeline is configured to execute hint calculus code that virtualizes instructions from the user program.

16. A processor comprising:

a conjugate mapping table coupled to a first pipeline, the conjugate mapping table including entries to map triggers from the first pipeline to hint calculus code sequences; and

a second pipeline to execute the hint calculus code sequences.

17. The processor of claim 16 further comprising microarchitectural structures responsive to the second pipeline, the microarchitectural structures being configured to modify states of the first pipeline.

18. The processor of claim 17 wherein the microarchitectural structures comprise a branch target buffer.

19. The processor of claim 17 wherein the triggers comprise:

instruction attributes;

data attributes;

state attributes; and

event attributes.

20. The processor of claim 16 further comprising:

an instruction cache coupled between the conjugate mapping table and the second pipeline, the instruction cache being configured to cache the hint calculus code sequences.

21. The processor of claim 16 wherein the processor is configured to execute essential code and non-essential code, and wherein the non-essential code comprises the hint calculus code sequences.

22. The processor of claim 16 wherein the second pipeline is configured to implement register file expansion for forward compatibility across different generations of processors.

23. The processor of claim 16 wherein the second pipeline is configured to implement built-in-self-test.

24. The processor of claim 16 wherein the second pipeline is configured to implement instruction set virtualization.

25. A computer-implemented method of creating a runtime binary comprising:

combining an essential portion of a static binary with an essential portion of a runtime library to create an essential portion of the runtime binary; and

creating a non-essential portion of the runtime binary using a non-essential portion of the static binary.

26. The computer-implemented method of claim 25 wherein creating a non-essential portion comprises:

combining the non-essential portion of the static binary with a non-essential portion of the runtime library.

27. The computer-implemented method of claim 26 wherein combining the non-essential portion comprises:

combining non-essential code from the static binary with non-essential code from the runtime library; and

combining conjugate mapping information from the static binary with conjugate mapping information from the runtime binary.

28. An article having a machine readable medium with instructions for performing a method of creating a runtime binary disposed thereon, the method comprising:

combining an essential portion of a static binary with an essential portion of a runtime library to create an essential portion of the runtime binary; and

creating a non-essential portion of the runtime binary using a non-essential portion of the static binary.

29. The article of claim 28 wherein creating a non-essential portion comprises:

combining the non-essential portion of the static binary with a non-essential portion of the runtime library.

30. The article of claim 29 wherein combining the non-essential portion comprises:

combining non-essential code from the static binary with non-essential code from the runtime library; and

combining conjugate mapping information from the static binary with conjugate mapping information from the runtime binary.